ASIP Designerで実現する RISC-Vプロセッサの開発と既存RTLを用いた機能拡張

RISC-V Days Tokyo 2021 Autumn

日本シノプシス合同会社
シニアアプリケーションエンジニア 伴野 充

2021年11月18日

アジェンダ

- シノプシスプロセッサソリューション
- ASIP Designerのご紹介
- RISC-Vプロセッサの開発
- 既存RTLを用いた機能拡張
- まとめ

Synopsys Today: シリコンからソフトウェアへ

FY20 売り上げ:
~\$3.7B

従業員数:
~15,000

取得特許:
~3,300

オフィス拠点:
~125

業界ナンバーワンの (EDA)電子設計
自動化ツールとサービス

幅広いIPポートフォリオと#1の
インターフェースIP、アナログ、
組込みメモリ、物理IPおよびプロセッサ

ガードナーのアプリケーションセキュリ
ティ・テスト向け“Magic Quadrant”の
リーダー

シノプシス・プロセッサ製品一覧

*開発中
 ASIL対応プロセッサ

ASIP Designerのご紹介

ドメインに特化したプロセッサの時代

Figure 1: Processor Performance vs. Time

Source: Xilinx Whitepaper 505 – Versal ACAP
(reference to J. Hennessy, D. Patterson, *Computer Architecture: A Quantitative Approach* (6th Edition, 2019))

- 1995 – 2003:
プロセス技術による性能(より高い周波数だが同じ電力消費)およびメモリアクセスの最適化(Dennard Scaling)
- 2003: Multicore
タスクの並列化制限に到達(Amdahlの法則)
- 2015: Domain-Specific Processors
ヘテロジニアス・マルチコア設計への移行

ASIP Designer

- アーキテクチャ構造記述言語(nML)により、
 - アプリケーションに特化した命令セットを持つ専用プロセッサ/DSPを自動生成する設計環境

ASIP設計開発ツール
“ASIP Designer”

Graph-Based C コンパイラ “Chess”

- フロントエンド
 - C → コントロールデータフローグラフ
 - nML → インストラクションセットグラフ
- コンパイルフェーズ
 - CDFGから ISGへマッピング
 - 独自“Graph”アルゴリズム
- ISGに含まれる構成情報
 - ハードウェアリソース
 - データタイプ
 - 接続情報
 - 命令エンコード情報
 - 命令レベルの並列化情報
 - 命令パイプライン情報

ASIP Designer - 命令セットシミュレータ

サイクル精度

↓ Pipeline View

Statistics Registers

PC	=	237	Loop regs	=	3
SP	=	77	LF	=	0
LR	=	0	LCD	=	0
R0	=	2	LC2	=	0
R1	=	3	SR	=	2
R2	=	374	SR	=	0
R3	=	9	SR	=	0
R4	=	-1	CND	=	0
R5	=	16	CD	=	1
R6	=	14			

Console

```
Elapsed time = 0.0219999998079071
#Cycles/(Elapsed) = 108545.455133619
```

命令精度

Local variables

Name	Type	Value	Location
Run_Index	int	374	DM[SP - 3]
void_Proc_5	void	374	R[2]
Str_1_Loc	char [31]	<array>	DM[SP - 45]
Int_1_Loc	int	5	DM[SP - 12]

Console

```
Elapsed time = 0.00400000189989805
#Cycles/(Elapsed) = 463499.97798493237
```

命令セット
シミュレータ

プロファイラ

Execution tracing

Function	Count
6	1257
5	1080
4	114
3	54
2	48
1	29
0	4

Function Execution Time

Id	Function name	Total func	% Total fu
6	motion_estimation void...	1257	48.8
5	sad_16x16_motion_sint...	1080	41.1
4	vfprintf_sint_vfprintf_P...	114	4.1
3	printf_sint_printf_P_cc...	54	2.0
2	_hosted_clib_io_hosted_cl...	48	1.8
1	main_main	29	1.1
0		4	0.1

Function summary:

Cycles	% of total	Instruction count	% of total	Coverage
1257	48.85%	1133	47.95%	100.00
1080	41.71%	1008	42.66%	100.00
114	4.40%	108	4.57%	100.00
54	2.08%	42	1.78%	100.00
48	1.85%	48	2.03%	100.00
29	1.12%	20	0.85%	90.91
4	0.15%	2	0.08%	100.00

Function detail: sad_16x16_motion_sint_sad_16x16_P_uc

PC	Instruction	Assembly
Low PC		4
High PC		18
Size in program memory:		15
Cycle-count:		1080 (41.71%)
Instruction-count:		1008 (42.66%)
Instruction Coverage:		86.67%

仮想プロトタイピング

Block name: micro_sc
Library: micro_sc
Instance name: I_AsipCore0
Encomp: micro_sc
Auto-Connect Alias: I_AsipCore0

ASIP Designer生成 - ASIPデバッガ

The screenshot displays the ASIP Designer IDE interface for a project named 'motion.prx - ChessDE'. The interface is divided into several panes:

- Project explorer:** Shows the project structure with files like 'tmotion.n' and 'motion.c'.
- Microcode pane (left):** Displays assembly code for functions like 'tmotion_init' and 'sad_16x16 __motion__ sint_sad_16x16 __P_ ucha'. A yellow box labeled 'アセンブラコード表示' (Assembly code display) points to this pane.
- Main editor pane (right):** Shows C code for 'motion.c'. A yellow box labeled 'nMLコード表示など' (nML code display, etc.) points to a line of code: `sad += vsumi(adiff);`. Below it, a comment block describes a step-based algorithm for motion estimation.
- Console pane (bottom):** A yellow box labeled 'メッセージウインドウ' (Message window) points to this area.

```
tmotion_init
0 241a 0002 mvi sp,2
2 2e80 007b j 123

sad_16x16 __motion__ sint_sad_16x16 __P_ ucha
4 2501 mv r0,r1
5 3204 mvib r4,32
6 c055 ld v1,dm(r1++)
7 2532 mv r3,r2
8 c061 ld v0,dm(r1+=r4)
9 aaa4 ld v1,dm(r1++) | ld v3,cm(r3++)
10 3008 mvib r6,0
11 2ee0 000f 0010 doi 18,15
14 b0b3 ld v0,dm(r1+=r4) | ld v3,cm(r3++)
15 2ee4 000f 0010 doi 18,15

motion_estimation void_motion_estimation __P_
19 5300 addb sp, 48
20 3003 mvib r3,0
21 7ec2 st r2,dm(sp-20)
22 7ee1 st r1,dm(sp-18)
23 7ea3 st r3,dm(sp-22)
24 7e83 st r3,dm(sp-24)
25 7e63 st r3,dm(sp-26)
26 7e40 st r0,dm(sp-28)
27 7e2b st l1,dm(sp-30)
28 3305 mvib r5,48
29 2417 7fff mvi r7,32767
```

```
int sad = 0;
vpix chess_storage(CMb)* pvc = (vpix chess_storage(CMb))*curr_block;
vpix* pvss = (vpix*)search_window;
vpix* pvs = pvss;
for (int i = 0; i < 16; i++) {
    vpix vc = *pvc++;
    vpix vs = valign(pvss,*pvs,*pvs+1));
    pvs += LINE_STEP/16;
    vpix adiff = vadiff(vs,vc);
    sad += vsumi(adiff);
}
return sad;

/* x and y steps define center point and 8 surrounding points
---
(8,-8) (8,0) (8,8)
(0,-8) (0,0) (0,8)
(-8,-8) (-8,0) (-8,8)
*/

static int x_steps[9] = { 0, 8, 8, 0, -8, -8, -8, 0, 8 };
static int y_steps[9] = { 0, 0, 8, 8, 8, 0, -8, -8, -8 };

struct Motion_vector { int x, y; };

void motion_estimation(unsigned char* search_window,
                      unsigned char chess_storage(CMb)* curr_block,
                      Motion_vector* v)
{
    // Compute motion vector using a step based algorithm with steps 8, 4, 2
    // and 1

    int sad, min_sad;
```

ASIP Designer生成 – ソフトウェアデバッガ

The screenshot shows the ASIP Designer software debugger interface. The main window is titled "motion.prx - ChessDE" and contains several panes:

- Microcode:** A list of microcode instructions with addresses and hex values. Line 152 is highlighted in green.
- Hosted I/O / Local variables:** A pane showing the current state of the program, including the C source code for "motion.c".
- Hosted I/O / Local variables:** A pane showing the output of the program, including the results of motion estimation steps.
- Console:** A pane showing the elapsed time and cycles per elapsed time.

Four callout boxes highlight specific features:

- アセンブラ表示 (Assembly Display):** Points to the Microcode pane.
- Cソースコード表示 (C Source Code Display):** Points to the C source code in the Hosted I/O / Local variables pane.
- レジスタ情報表示 (Register Information Display):** Points to the Registers pane at the bottom left.
- メッセージウインドウ (Message Window):** Points to the Console pane at the bottom right.

ソフトウェア・プロファイリング機能

命令プロファイリング

Function summary:

Cycles	% of total	Instruction	% of total	Coverage
1620	51.01%	1548	52.80%	86.67%
1258	39.81%	1133	38.50%	90.38%
131	4.12%	124	4.21%	94.74%
63	1.98%	49	1.66%	77.78%
56	1.78%	56	1.90%	100.00%
38	1.20%	27	0.92%	70.97%
7	0.22%	4	0.14%	75.00%

Function detail: sad 16x16 motion sint sad 16x16 P u

Low PC : 4
High PC : 18
Size in program memory : 15
Cycle-count : 1620 (51.01%)
Instruction-count : 1548 (52.80%)
Instruction Coverage : 86.67%

PC	Instruction	Assembly	Exe-count	Cycles	Wait states	Relative cycle use within func
4 2501	mv r0,r1		36	36	0	***
5 3204	mvib r4,32		36	36	0	***
6 c055	ld v1,dm(r1++)		36	36	0	***
7 2532	mv r3,r2		36	36	0	***
8 c061	ld v0,dm(r1+r4)		36	36	0	***
9 aaa4	ld v1,dm(r1++) ld v3,cm(r3++)		36	36	0	***
10 3006	mvib r8,0		36	36	0	***
11 2ee0 000f 0010	doi 16,15		36	108	0	*****
14 b0b3	ld v0,dm(r1+r4) ld v3,cm(r3)		576	576	0	*****
15 aac4	ld v1,dm(r1++) add r6,r5,r6		576	576	0	*****
16 2ec0	rti		36	36	0	***
17 2506	mv r0,r6		36	36	0	***
18 2e00	nop		36	36	0	***

Function detail: motion_estimation void_motion_estimation_P

Low PC : 19
High PC : 122
Size in program memory : 104
Cycle-count : 1258 (39.81%)
Instruction-count : 1133 (38.50%)
Instruction Coverage : 90.38%

命令 & サイクル・カウント

```
ISS command = <PROCDIR>/../iss/tmotion_ca
ISS mode    = Cycle accurate
```

Cycle count = 3176
Instruction count = 2943

PC = 130

SP = 98
Stack area: DMb[2 .. 2049], growing up
Maximum stack pointer value = 146

Instruction history:

Stg	PC	Instruction	Assembly
IF	130	----	----
ID	180 5c00	----	addb sp, -64
E1	179 6d00	----	ld r0,dm(sp-48)
S3	178 2ec0	----	rti
S4	177 6feb	----	ld lr,dm(sp-2)
S5	161 2e00	----	nop
S6	160 4201	----	st r1,dm(r0)
S7	159 2ec0	----	rti
S8	158 3101	----	mvib r1,16
S9	157 2e00	----	nop
S10	156 2e00	----	nop

命令 & サイクル・チャート

パイプラインビュー

nMLカバレッジ

```
tmotion (0.266224)
.alu_instr (0.0654762)
.alu_rrr (0.142857)
+alu_op (0.142857)
-add (1)
-addc (0)
-sub (0)
-subb (0)
-and (0)
-or (0)
-xor (0)
.compare_rr (0.25)
+compare_op (0.25)
-it (1)
-itu (0)
-le (0)
-leu (0)
-gt (0)
-gtu (0)
-ge (1)
-geu (0)
.equal_rr (0)
.alu_rr (0)
.minmax_rrr (0)
.select_rrr (0)
.shift_instr (0.333333)
.shift_rrr (0.333333)
```

RTLコード生成と実機デバッグ環境の構築

マルチASIPデバッグが可能

- nML(PDG)記述からRTLコードを生成
 - Verilog/VHDL RTLコード
 - JTAG Key/JTAG key2をサポート
 - Design Compiler/ Simplify用スクリプト
 - 生成プロセッサ(RTL)のロイヤリティ不要

RISC-Vプロセッサの開発

ASIP Designer

- アーキテクチャ構造記述言語(nML)により、
 - アプリケーションに特化した命令セットを持つ専用プロセッサ/ DSP を自動生成する設計環境

ASIP設計開発ツール
“ASIP Designer”

命令セット
シミュレータ(ISS)

C/C++ コンパイラ
アセンブラ/リンカー

デバッガ/プロファイラ

Verilog / VHDL
RTL コード

ご提供可能なExampleプロセッサ

種類	説明
マイクロコントローラ	
Tnano	16-bit microcontroller, lightweight and configurable
Tmicro	16-bit microcontroller, fully featured
DLX (family)	Variants of 32-bit microcontroller
Tmcu	32-bit microcontroller
Trv32 (family)	32-bit microcontroller featuring RISC-V ISA, 3 or 5 pipeline stages
Trv64 (family)	64-bit microcontroller featuring RISC-V ISA, 3 or 5 pipeline stages
PD_Triop	32-bit microcontroller with 64-bit address spaces
DSP及び並列演算エンジン	
Tdsp	16/32-bit DSP
Tvec (family)	Variants of SIMD processor
Tvliw (family)	Variants of VLIW processor
暗号処理エンジン	
Tsec	High throughput SHA/RSA/AES accelerator
Tcript	High throughput AES accelerator
アプリケーション特化	
Tmoby	Accelerator for AI application, featuring MobileNet V3 graph
Tvox	Accelerator for SLAM (simultaneous localization and mapping)
MMSE	Minimum Mean Square Error Equalization
Tgauss	Vectorization and memory management for image processing
Tmotion	Accelerator of motion estimation kernel
Tcom8	SIMD processor optimized for some communication kernels
FFTcore	Scalar implementation of complex FFT

green =
released in 2020

nML記述
ソースコード
ご提供

豊富なExample

- マイクロコントローラ
- DSPs
- SIMD、VLIW
- マルチスレッド
- セキュリティ(AES、SHA等)
- メモリ/F、AXIバス等

Example使用のメリット

- 基本的な機能は既に設計開発済み
- 動作可能な例を参照可能
- モデリングの概念を効率的に習得
- nMLソースコードで提供されるため、ユーザーが自由に追加変更可能

Trv(RISC-V ISA)モデル

内容

- RISC-V ISAに最適化されたモデル
- ISA拡張のスターティングポイントとしての活用 (ASIP設計)

	32-bit データパス	64-bit データパス
3ステージ パイプライン	Trv32p3 Trv32p3x	Trv64p3 Trv64p3x
5ステージ パイプライン	Trv32p5 Trv32p5x	Trv64p5 Trv64p5x

機能 (2020.03)

- 対応ISA: RV64IM, RV32IM
 - 整数命令 (Integer)
 - 乗算命令 (Multiply)
- オプション拡張: Trv<mm>p<n>x
 - (標準) 圧縮命令 (Compressed: IMC)
 - (カスタム) ゼロオーバーヘッドループ
 - (カスタム) アドレス更新後のロード/ストア
- マイクロアーキテクチャの特長
 - 5段ステージパイプライン: IF, ID, EX, ME, WB
 - 3段ステージパイプライン: IF, ID, EX
 - 保護されたパイプライン (Protected pipeline)
 - 可能な場合はレジスタをバイパス
 - バイパスが出来ない時はStall (bubble)
 - ハードウェア乗算器
 - 64x64 → 128 → select 64bit,
 - 32x32 → 64 → select 32bit
 - 繰り返し処理の除算器

▲ Trv<mm>p3 datapath and pipeline

▲ Trv<mm>p5 datapath and pipeline

Trvモデル拡張 (2021.03)

F-クラスサポート, オンチップデバッグ

- Trv32モデルにオプションのF命令クラスを拡張
F命令クラス

- IEEE754 単精度浮動小数点のサポート
(round to nearest-even mode)
- レジスタファイル
- FADD, FSUB, FMUL, FCMP (EQ/LT/LE), FMIN, FMAX, FCVT → DesignWareモジュール
- FDIV, FSQRT → 繰り返し処理の実装

ISA	28nm TSMC	
	HS (maxfreq)	HD (minarea)
Trv32p5f (IMF)	1.37 GHz	50MHz (target)
	75.6 kGE	54.7 kGE

- Trvモデルにオプションのオンチップデバッグを拡張(OCD)

- ASIP DesignerのOCD自動生成機能を使用
- 自動的にISAの拡張に適応

SDX: シンプルデータパス拡張

- SDXは(RISC-V)プロセッサモデルに単純な拡張命令を追加するメカニズム
 - 拡張命令スケルトンは、プロセッサモデルの一部として提供
 - ユーザーは特別なnML記述を学ぶ必要がない
 - ユーザーが拡張機能の動作をPDGとしてコーディング
 - PDGは、データパス関数の動作を定義するために使用されるCのような言語
 - 拡張命令をターゲットとするコンパイラ組み込み関数を提供
- RISC-Vの“custom-2”命令としてエンコード(後述)
- 命令バリエーション
 - 3つのreg 32ビットおよび64ビット、累積、追加のシングルレジスタ入力および出力等々, ...
- 例: SHA256 (オペレーション融合), FFT (コンプレックス処理), CNN (パッケージ化されたSIMD)

Behavioral model (PDG)

```
w32 sdx0(w32 a, w32 b)
{
  w32 r;
  r[15:0] = a[15:0] + b[15:0];
  r[31:16] = a[31:16] + b[31:16];
  return r;
}
```

Compiler intrinsics

```
int sdx0(int,int);
int add2(int,int);
```

既存RTLを用いた機能拡張

ASIP DesignerにおけるRTL生成

- 合成可能なVHDL/Verilogコードを生成
- 生成されたコードの構成
 - トップレベルネットリスト
 - ファンクショナルユニット(FU)、レジスタ、およびその他のモジュール
 - MUXベースのインターコネクト
 - モジュールのビヘイビアモデル
- FUのビヘイビアモデルはPDG言語で記述
 - 例: add with status output

```
word add(word a, word b, astat& s)
{
  int17_t aa = (uint16_t)a;
  int17_t bb = (uint16_t)b;
  int17_t d = aa + bb;
  s = (d[16]^d[15]) :: (d<0) :: (d==0);
  return d;
}
```

ケース1: 既存RTLによる組合せ回路の実装

FLX

- ハードウェアによる浮動小数点をサポートした32-bit RISC
- 異なるFPU実装による3つの派生形

FLX1: PDGベースモデル

- IEEE単精度浮動小数点を実装
- PDGモデルを使用
 - ISSモデル
 - RTL生成

FLX2: DesignWareベースのRTL

- RTL実装: DWインスタンスを持つラッパー
- ISSモデル: C++ビルトイン浮動小数点加算

FLX3: Non-IEEEフォーマット

```
class float24_t
property(
24 bits,
sign_magnitude,
8 exp_bits);
```

- RTL: DesignWare
- ISS & Native
 - リファレンスのVerilogモデルをVerilatorツールでC++に変換

ケース2: 既存RTLによる繰返し操作の実装

既存のVerilog RTLモジュールとISSを用いてマルチサイクル除算を実装

- 除算はマルチサイクルのプリミティブとしてモデル化
- マルチサイクルファンクショナルユニットを使用して、
 - Start信号をドライブして、オペランドを出力ポートdiv_opAと div_opBにコピー
 - ready信号を確認して、結果をライトバック
 - パイプライン制御用の信号を生成
- IOインターフェースをRTLモジュールに対するプレースホルダーとして使用される
 - GoツールがVerilogファイルにあるuser_div.vをIOインターフェースモジュールの代わりにコピー
- IOインターフェースはISSに対してuser_div C++クラスを生成するためにも使用される
 - コンストラクタ user_div(dlx*)
 - メンバ main()
 - ステータスアップデート update_status()
 - Verilatorツールによってuser_div.vをビットアキュレートかつサイクルアキュレートなC++コードに変換

既存RTLコードのインポートフロー

PDGで定義されていない
ダミーのファンクショナル
ユニットを持つnMLモデル

初回のGo
ツールの実行

空のファンクショナルユニット
を持つ生成されたRTL

2回目のGo
ツールの実行、
既存RTLをイン
ポート

インポートされたファンクショナル
ユニットを持つ再生成されたRTL

ラッパーの作成

- ① 空モジュールを持つファイルをコピー
- ② 既存RTLコードを統合
- ③ 編集したモジュールをGoツールのコンフィギュレーションオプションでインポート

まとめ

ASIP Designer – ASIP デザインを容易に

- アプリケーション特化によるPPAの大幅な改善！
- RISC-V ISAベースのExampleモデルで即座にトライアルが可能！
- 既存の設計資産(RTL)を即座に流用可能！
- **製品の差別化を実現する拡張可能かつロイヤリティフリーなRISC-Vプロセッサを開発！**
 - ご興味のある方はぜひご連絡ください
 - コンタクト先: jp-mkg_info@synopsys.com

Thank You